

**LINK***group*

Distance Learning System

$f_x$

# Funkcije

Python and programming fundamentals

# Šta je funkcija

---

- **Funkcija** je zasebna celina u programu koja se može izvršiti na zahtev nekog drugog dela programa
- Funkcija olakšava kodiranje i smanjuje količinu ponovljenog koda
- Funkcije mogu biti **ugrađene** ili **korisnički definisane**
- Funkcije mogu postojati samostalno ili u okviru nekih objekata, kada se nazivaju **metode**

# Gde se koriste funkcije

---

- Funkcije se koriste uvek i svugde

# Kreiranje funkcije

---

- Ključna reč za kreiranje funkcije je **def**
- **Naziv funkcije** može biti proizvoljan, ali mora poštovati pravila imenovanja promenljivih
- **Male zagrade** iza naziva funkcije su obavezne i mogu biti prazne ili sadržati ulazne parametre funkcije
- **Dvotačka** označava početak tela funkcije
- Telo funkcije sadrži linije sa **uvlakom**

**Potpis funkcije** | `def myfirstfunction():`

**Telo funkcije** | `→ print("Hello from function!")`

# Pozivanje funkcije

---

- Funkcija se poziva **nazivom funkcije** kome slede **male zagrade**

```
myfirstfunction()
```



```
Hello from function!
```

# Ulazni parametri funkcije

---

- Funkcija može prihvatiti parametre prilikom startovanja
- Parametri se označavaju unutar malih zagrada, u potpisu funkcije

```
def myfirstfunction(msg):  
    print(msg)
```

- Ovako parametrizovana funkcija, poziva se takođe parametrizovano

```
myfirstfunction("Hello")
```

- Poziv bez parametara nije dozvoljen i izaziva grešku

```
myfirstfunction()
```

```
TypeError: myfunction() missing 1 required positional argument: 'msg'
```

# Podrazumevani parametri funkcije

- Podrazumevani parametri su parametri koji se mogu, ali ne moraju proslediti funkciji
- Ovakvi parametri moraju biti posebno označeni tako što će im biti dodeljena podrazumevana vrednost

```
def myfirstfunction(msg="No message") :  
    print(msg)
```

- Ovakom definisana funkcija, može biti pozvana sa ili bez parametara

```
myfirstfunction("Hello")
```



Hello

```
myfirstfunction()
```



No message

# Imenovani parametri funkcije

- Prilikom pozivanja, parametre je moguće prosleđivati koristeći njihov redosled u potpisu funkcije, ali takođe i koristeći njihove nazive

```
def myfirstfunction(msg="No message", msg1="No message"):  
    print(msg)
```

```
myfirstfunction(msg="Hello", msg1="World")
```

```
myfirstfunction("World", "World")
```



Hello



# Izlazni parametri funkcije

- Osim što ih može prihvatiti, funkcija takođe može i vratiti vrednost. Ova vrednost najčešće predstavlja rezultat rada funkcije (neke računske operacije, dobavljanje podatka iz nekog izvora, intervencija na nekom podatku i slično)
- Vrednost funkcije se vraća ključnom rečju **return**, kojoj sledi vrednost koja treba da bude vraćena

```
def myfirstfunction():  
    return ("Hello")
```

- Funkcija koja vraća vrednost, tretira se u programu kao konstanta

```
res = myfirstfunction() ← Ovo je isto kao da  
print(res)                smo napisali Hello
```

# Izlazni parametri funkcije

---

- Funkcija može vratiti i više vrednosti
- U slučaju da funkcija vraća više vrednosti, možemo ih preuzeti pomoću više promenljivih, odvojenih zarezima

```
def myfunction():  
    return 1,2,3  
  
a,b,c = myfunction()  
print(a,b,c)
```

# Proizvoljan broj parametara (varargs)

---

- Funkcije se moraju pozivati sa očekivanim brojem parametara
- Dva specijalna operatora (\* i \*\*) omogućavaju da se funkcija pozove sa proizvoljnim brojem i nazivima parametara
- Svi parametri prosleđeni funkciji biće pretvoreni u niz

```
def f(*args):  
    print(args[0] + args[1])
```

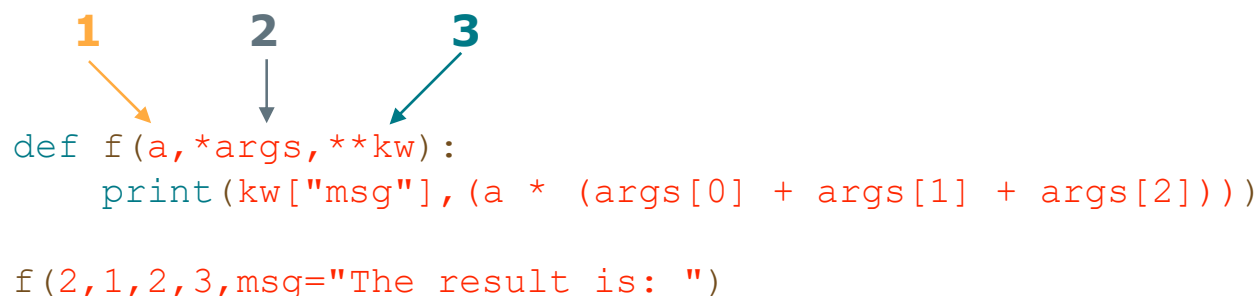
```
f(2, 3)
```

```
def f(**args):  
    print(args["firstname"])
```

```
f(firstname="Peter")
```

# Proizvoljan broj parametara (varargs)

- Varargs se mogu koristiti u kombinaciji, kao i u kombinaciji sa obaveznim parametrima
- Redosled je važan:
  1. Obavezni pozicioni parametri
  2. Varargs pozicioni parametri
  3. Varargs imenovani parametri



```
def f(a, *args, **kw):  
    print(kw["msg"], (a * (args[0] + args[1] + args[2])))  
  
f(2, 1, 2, 3, msg="The result is: ")
```

# Vežba Kalkulator (ppf-ex09 calc.py)

---

- Kreirati funkciju calc
- Funkcija prihvata tri parametra: dva broja i operaciju (operacija u formi teksta)
- Funkcija vraća rezultat operacije nad dva uneta broja

```
print(calc(2,3,"mul"))
```

6

# Vežba Render (ppf-ex09 render.py)

---

- Kreirati funkciju koja iscrtava prozor, na osnovu unete širine i visine

`render(10,10)`

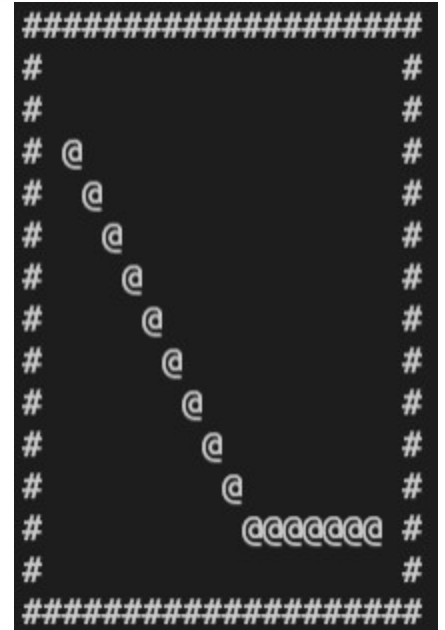


```
#####  
#           #  
#           #  
#           #  
#           #  
#           #  
#           #  
#           #  
#           #  
#           #  
#####
```

# Vežba My way (ppf-ex09 myway.py)

- Modifikovati funkciju iz vežbe render, tako da prihvata i treći parametar sa listom tačaka za crtanje
- Kreirati funkciju koja vraća listu tačaka na osnovu početne i krajnje tačke
- Niz tačaka treba da sadrži putanju od početne i krajnje tačke

```
render(20,15,path([2,3],[17,12]))
```

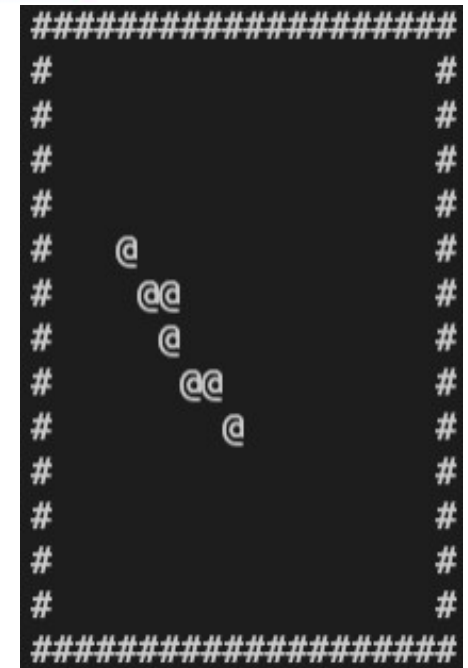


# Vežba Update (ppf-ex09 update.py)

- U programu je definisana početna tačka i brzina
- Potrebno je kreirati funkciju update, koja kao parametar prihvata krajnju tačku i izračunava sledeću tačku na putu ka krajnjoj tački
- Potrebno je kreirati funkciju, koja pomoću funkcije update, generiše listu tačaka koje čine putanju od početne do krajnje tačke

```
render(width,height,path(4,5))
```

- Formule:
  - x i y od ugla i brzine:
$$x += speed * \cos(an)$$
$$y += speed * \sin(an)$$
  - Ugao od x,y i x1,y1:
$$an = \text{atan2}(y1 - y, x1 - x)$$





# Ugrađene funkcije

<https://docs.python.org/3/library/functions.html>

- Prilikom aktivacije, Python interpreter ima na raspolaganju određene ugrađene funkcije, koje se mogu koristiti odmah
- Neke od ovih funkcija su već **obrađene** u prethodnim lekcijama, neke su **dovoljno intuitivne** da se i ne moraju posebno objašnjavati

<code>abs()</code>	<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>
<code>all()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>any()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>ascii()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>bin()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bool()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>breakpoint()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	

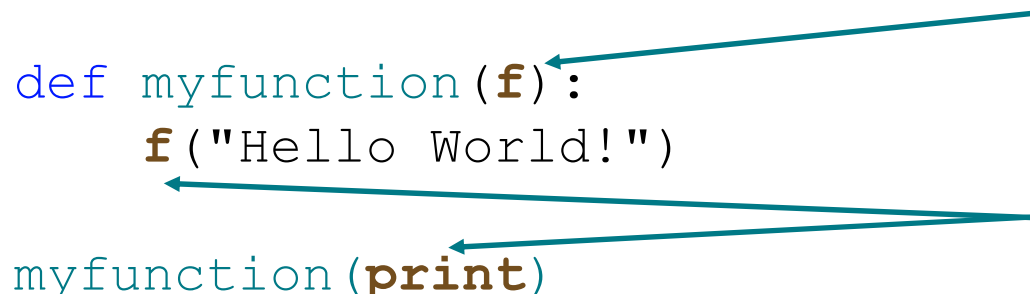
# Prosleđivanje funkcije

---

- U nekim situacijama, želimo da, kao parametar funkciji, prosledimo samu funkciju
- Ovaj koncept se često koristi u različitim situacijama (upravljanje događajima, publikovanje poruka i slično)

```
def myfunction(f):  
    f("Hello World!")  
  
myfunction(print)
```

Ugrađena funkcija **print**, prosleđena je kao parametar korisnički definisanoj funkciji **myfunction**



# Anonimna funkcija - lambda

---

- Nekada želimo da umesto prosleđivanja postojeće funkcije kreiramo funkciju koju prosleđujemo u trenutku prosleđivanja
- U takvim situacijama, koriste se lambda izrazi
- Lambda se kreira na sledeći način

Ključnoj reči **lambda** sledi **lista parametara**, **dvotačka** (**ne mora biti ni jedan ulazni parametar**) i **izraz** koji predstavlja povratnu vrednost

```
z = lambda : 10  
print(z())
```

# Rekurzivna funkcija

- Rekurzivna funkcija je funkcija koja poziva samu sebe

```
def f(i):  
    print(i)  
    i-=1  
    if i > 0:  
        f(i)  
f(10)
```



```
10  
9  
8  
7  
6  
5  
4  
3  
2  
1
```

- Kod rekurzivne funkcije je veoma važno imati neku vrstu kontrole, koja će onemogućiti beskonačno ponavljanje poziva

# Dekoratori

---

- Dekorator je tehnika kojom možemo modifikovati ponašanje funkcije tokom izvršavanja
- Dekorator podrazumeva funkciju koja vrši modifikaciju i funkciju nad kojom se vrši modifikacija
- Funkcija koja vrši modifikaciju je obična funkcija
- Funkcija nad kojom se vrši modifikacija, označava se oznakom @
- Dekoratori su primenjivi i na druge strukture (klase i metode)

# Kreiranje i upotreba dekoratora

- Kao dekorator može biti upotrebljena bilo koja funkcija koja kao parametar prima funkciju i kao povratnu vrednost vraća funkciju

```
def mydecorator(f):  
    #do something here  
    return f
```

Ovo će biti funkcija



- Funkciju koju nameravamo da koristimo kao dekorator, primenjujemo pomoću oznake @ nad funkcijom koju dekorišemo

```
@mydecorator  
def myfunction(a,b):  
    return a+b
```

# Kreiranje i upotreba dekoratora

---

- Osim presretanja dekorisane funkcije, takođe je moguće u potpunosti prepisati njen sadržaj

```
def mydecorator(f):  
    def rewritten(*args,**kw):  
        return ("Result is " + str(f(*args)))  
    return rewritten
```

```
@mydecorator  
def myfunc(a,b):  
    return a + b  
  
print(myfunc(2,3))
```

—————→ Result is 5

# Generatori

- Generator omogućava parcijalno, sekvencijalno izvršavanje delova funkcije

```
def f():  
    print("Hello")  
    yield  
    print("World")  
    yield
```

`x = f()`

`next(x)`

`next(x)`

Još uvek nema izlaza

Hello

World



# Zašto bismo koristili Generator

- Generator omogućava jednostavnu implementaciju **iteratora**
- Ukoliko se komanda **yield** koristi sa parametrom ona izlaže sledeći element u iteratoru
- Generator je koristan ako želimo da implementiramo sopstvenu logiku za **iteraciju**

```
def f():  
    yield "Marco"  
    yield "Polo"
```

```
for i in f():  
    print(i)
```

→ Marco  
Polo

```
def counttoten(odd):  
    yield 1 if odd else 2  
    yield 3 if odd else 4  
    yield 5 if odd else 6  
    yield 7 if odd else 8  
    yield 9 if odd else 10
```

```
print("Odds: ")  
for i in counttoten(True):  
    print(i)  
  
print("Evens: ")  
for i in counttoten(False):  
    print(i)
```

→ Odds:  
1  
3  
5  
7  
9  
Evens:  
2  
4  
6  
8  
10